AFRL-HE-WP-TP-2007-0006

Thinking Opposing Force (OPFOR) for Joint Conflict and Tactical Simulation (JCATS)

Randy Jones

Soar Technology, Inc.
3600 Green Court, Suite 600
Ann Arbor MI 48105

August 2003

Interim Report for August 2002 – August 2003

Air Force Research Laboratory
Human Effectiveness Directorate
Warfighter Interface Division
Cognitive Systems Branch
Wright-Patterson AFB OH 45433-7604

NOTICE

AFRL-HE-WP-TP-2007-0006

HAS BEEN REVIEWED AND IS APPROVED FOR PUBLICATION IN ACCORDANCE WITH ASSIGNED DISTRIBUTION STATEMENT.

FOR THE DIRECTOR

//SIGNED//

DANIEL G. GODDARD
Chief, Warfighter Interface Division
Human Effectiveness Directorate
Air Force Research Laboratory

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From - To)*
August 2003	Interim	August 2002 - August 2003

4. TITLE AND SUBTITLE
Thinking Opposing Force (OPFOR) for Joint Conflict and Tactical Simulation (JCATS)

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER
63832D

6. AUTHOR(S)
Randy Jones

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER
0476DM00

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Soar Technology, Inc.
3600 Green Court, Suite 600
Ann Arbor MI 48105

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Air Force Materiel Command
Air Force Research Laboratory
Human Effectiveness Directorate
Warfighter Interface Division
Cognitive Systems Branch
Wright-Patterson AFB OH 45433-7604

Defense Modeling and Simulation Office
1901 N. Beauregard Street, Suite 500
Alexandria VA 22311-1705

10. SPONSOR/MONITOR'S ACRONYM(S)
AFRL/HECS, DMSO

11. SPONSOR/MONITOR'S REPORT NUMBER(S)
AFRL-HE-WP-TP-2007-0006

12. DISTRIBUTION / AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited.
AFRL/PA cleared on 02 July 2007, AFRL-07-1552.

13. SUPPLEMENTARY NOTES

14. ABSTRACT
The project discussed in this report focused on two primary research objectives. The first objective was to evaluate the technical feasibility and costs associated with introducing autonomous human behavior models into the Joint Conflict and Tactical Simulation (JCATS) environment. The second objective was to evaluate and recommend improved graphical user interfaces for specifying JCATS entity behaviors, which would aid both scenario generation and execution. This project developed a demonstration scenario for a "Thinking OPFOR" (Opposing Force) capability for entities in a JCATS Military Operations in Urban Terrain (MOUT) scenario, which drove an analysis of the ability to provide "thinking" type entities and deploying such entities within the JCATS infrastructure. This analysis was used to propose two alternative solution paths to providing realistic human-like behaviors for JCATS entities. For the second objective, a prototype behavior editor was designed and built to specify autonomous entity behaviors within the current JCATS infrastructure. This prototype was used to analyze design recommendations for alternative types of user interfaces for future JCATS tools.

15. SUBJECT TERMS Human Behavior Modeling, Joint Conflict and Tactical Simulation (JCATS), JCATS Entity Behaviors, Synthetic Entities, Opposing Force (OPFOR)

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			John L. Camp
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	SAR	28	19b. TELEPHONE NUMBER *(include area code)*

Standard Form 298 (Rev. 8-98)
Prescribed by ANSI Std. 239.18

THIS PAGE LEFT INTENTIONALLY BLANK

Table of Contents

THIS PAGE LEFT INTENTIONALLY BLANK

1.0 Introduction

Joint Conflict and Tactical Simulation (JCATS) is a popular simulation system used for training and experimentation by various elements of the US government, including the Departments of Defense, Energy, and Treasury. JCATS provides a portable and high-fidelity way to simulate terrain effects and large numbers of entities participating in interactive scenarios. However, such scenarios typically require a great deal of manpower to generate and to execute. Scenario generation involves detailed specification of force profiles and rudimentary behaviors for a large number of entities participating in the simulation. Scenario execution (except for very small scenarios) generally requires the use of a number of human operators to manipulate and control the flow of the scenario, particularly by moving around entities and making them behave in natural ways.

The goal of our project was to investigate particular ways of reducing the expense involved in generating and executing JCATS scenarios. The specific approach we adopted relied on a number of assumptions about applications of JCATS:

1. JCATS scenarios require the inclusion of synthetic entities representing opposing force (OPFOR) and other forces. This requirement exists for most typical training and experimentation simulations.
2. Behaviors for OPFOR and other entities in JCATS are typically generated by human operators interacting with graphical user interfaces for controlling the movement and other specific actions of the entities.
3. The result of the required human involvement increases the expense of using JCATS for training and experimentation simulations. This expense comes in the form of manpower costs in terms of training and using human participants, as well as time costs for using humans who might better be spending their time on other tasks. These expenses ultimately boil down to real dollar costs for the time and expense of training human controllers and taking advantage of their time to generate and execute simulation scenarios.
4. Autonomous software entities can reduce the manpower requirements associated with running simulations, consequently reducing their expense.
5. Such entities must generate behaviors that are sufficiently realistic to provide positive training and experimentation results.
6. To minimize expense and ease use, such entities must be understandable, constructible, and maintainable by human trainers, experimenters, and operations planners.

2.0 Objectives

Consequent to our assumptions and goals, this project focused on two primary research objectives. The first objective was to evaluate the technical feasibility and costs associated with introducing autonomous human behavior models into the JCATS simulation environment. The second objective was to evaluate and recommend improved graphical user interfaces for specifying JCATS entity behaviors, which would aid both scenario generation and execution. In pursuit of the first objective, we designed a demonstration scenario within JCATS that would demonstrate a "Thinking OPFOR" capability for entities in a JCATS military operations in urban terrain (MOUT) scenario.

We built the demonstration scenario within the existing JCATS infrastructure, without creating any additional supporting tools, in order to evaluate JCATS' current abilities do provide entities with such autonomous behaviors. We used this scenario to drive an analysis of the JCATS system's ability to provide "thinking" type entities, evaluating the technical challenges of building and deploying such entities within the current infrastructure. We used this analysis to propose two alternative solution paths to providing realistic human-like behaviors for JCATS entities. In pursuit of the second objective, we used human-computer interaction principles to design and build a demonstration prototype of a behavior editor for specifying autonomous entity behaviors within the current JCATS infrastructure. We also used this prototype to analyze design recommendations for alternative types of user interfaces for future JCATS tools.

Further refining our research objectives, we pursued the following specific technical objectives in the service of evaluating autonomous entity technology and supporting user interfaces within JCATS:

1. Determine how best to simplify scenario generation for JCATS operators, using a combination of intelligent entities and improved user interfaces.
2. Create a small set of sample intelligent behaviors within the existing JCATS behavior infrastructure, together with supporting design methodologies for developing such behaviors, and documentation to support future development of similar behaviors.
3. Generate prototypes of autonomous agents and improved behavior-specification interfaces, together with recommendations for future enhancements to the JCATS simulation environment, in order to support easier deployment of autonomous agents and their supporting user interfaces.

3.0 Defining the Objectives

This project centers on adding a capability to generate "thinking" synthetic entities within JCATS. Thus, it is prudent to define which specific entity capabilities and properties we intend. For our purposes, a "thinking" entity has the following properties:

1. It is capable of goal-directed behavior. With this capability, entities can be instructed with high-level goals, such as "defend this building" or "flank the enemy", and the entities have access to sufficient knowledge to reasonably attempt to achieve those goals without further direction (unless further direction would be warranted).
2. It is capable of reactive behavior. Within the context of the specified goals, the agent can adapt and react to changes in the environment in order to react flexibly in ways that will still achieve the goals even in the face of changing constraints. For example, a thinking entity will autonomously determine when it is appropriate to take cover or return fire.
3. It is capable of maintaining human-like situational awareness. In support of an appropriate mixture of goal-driven and reactive behavior, the entity maintains data structure that represent a (potentially complex) model of the current state of the world, and conditions affecting and constraining the entity's goals. Maintaining such situational awareness allows the entity to manage realistic beliefs about the world around it, and to make effective interpretations of the

environment based on its goals, environment, mission parameters, and other relevant factors.

4.0 Enhancing the Behavior of JCATS Entities

One of our primary activities in this project was to assess two major alternatives to enhancing the JCATS system's ability to support ease development and deployment of thinking entities. This effort focused on alternatives for representing and editing entity behaviors. The first alternative considered aimed at working as much as possible within current JCATS structures. This effort involved examining the current behavior editor and capabilities within JCATS, and making recommendations for how the JCATS behavior infrastructure would need to be changed in order best to support thinking entities. This analysis focused on improving sensor and effector models that would allow the entities unrestricted movement in the simulated terrain, and would facilitate coordination and communication between autonomous synthetic teammates. Additionally, the analysis examined the incorporation into the JCATS behavior engine of data structures explicitly geared toward supporting the internal representation of complex goal interactions and situational awareness. The primary technical challenge to this alternative involves managing the potential expense and complexity of introducing a new behavior paradigm into the existing software structures in JCATS. This would have the advantage of maintaining the uniform development architecture for JCATS, but would possibly be technically infeasible if software interactions demanded a number of changing to existing JCATS code. This alternative would also run the risk of introducing new bugs into existing code in places where new enhancements are necessary.

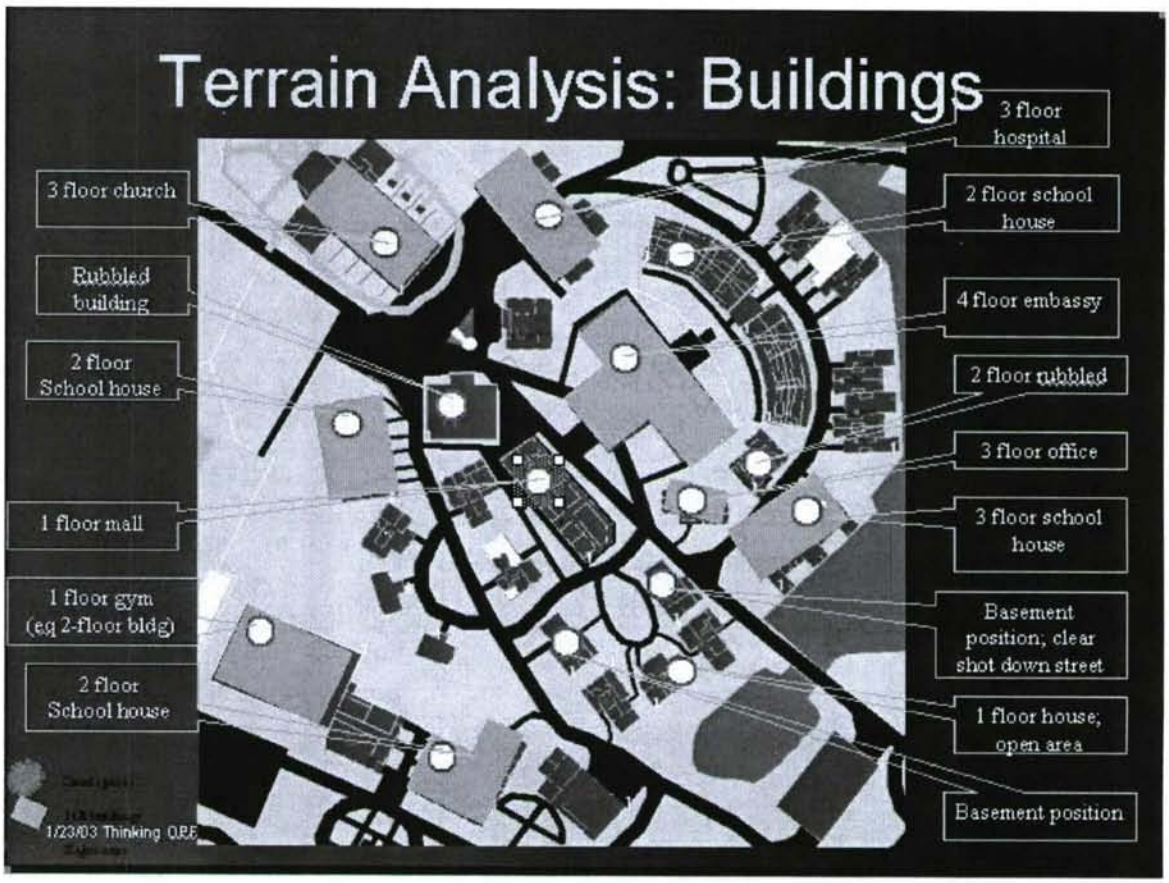
The second major alternative we considered for supporting thinking entities was to stay within the JCATS networking protocols and environment, but develop a behavior engine that stands outside the existing JCATS client and server software. This approach implements a distributed model of entity behavior, where autonomous behaviors are generated by an external "behavior server". This distinct piece of software would have at its core an existing software architecture for human behavior models, which would already include structures and processes for goal-directed reasoning, reactive reasoning, and representing situational awareness. The behavior server would add to this core engine network interfaces that are able to communicate to JCATS servers and clients via the existing JCATS networking protocols. The external network interface would include messages sending behavior data from the behavior server to individual JCATS clients and servers, as well as messages sending sensory and environmental information from the JCATS server to the behavior server. The primary technical challenge here is to ensure that the JCATS network protocol supports all the types of information interchange that would be required by a behavior server. However, most of this information must be provided to JCATS clients for display purposes anyway, so the amount of extra information necessary is likely to be small. A large advantage of this approach is that the networking code can be adjusted independently of the rest of the JCATS simulation, greatly reducing the risk of introducing any new bugs into existing software.

In order to assess these alternatives, we began by focusing on the specific technical changes that would be required to pursue alternative one, building an enhanced behavior capability into the existing JCATS framework. In order to identify and assess these changes, we created a prototype "Thinking OPFOR" demonstration scenario that

would include entities with a level of autonomy not usually used in JCATS scenarios. While constructing the scenario and the behaviors of the entities, we recorded and documented technical obstacles we encountered, and developed short-term solutions and long-term plans for dealing with each. We also assessed the current JCATS documentation and user interfaces for creating and editing entity behaviors. After constructing the demonstration scenario and fully describing obstacles and potential solutions, we held a meeting with the JCATS development team to assess the technical feasibility of implementing each proposed solution, and compared that with the feasibility and expected costs of pursuing alternative two (networking a behavior server with JCATS).

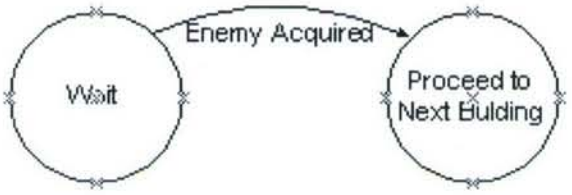
5.0 Thinking OPFOR Demonstration Scenario

We designed the demonstration scenario in order to highlight particular types of autonomous entity behavior, together with the hope of building a scenario and set of behaviors that could be used beyond this project (for example, to see actual use in JCATS training scenarios, or to serve as templates for other operators constructing scenarios). The scenario is based on a "Delay" type mission performed by OPFOR terrorists occupying building in an urban terrain setting. As Blue team members approach two defensive positions, OPFOR will attempt to delay and disperse them. We constructed the scenario with the help of Alion JCATS experts and analysts. Initial scenario design used terrain analysis to determine the best placement for the OPFOR.

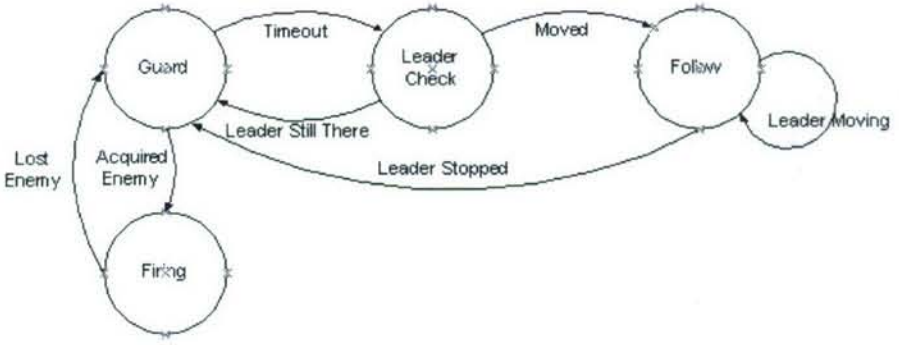


OPFOR strategies include sniper placement on various building, plus trigger lines for delay and retreat behaviors. Blue forces have pre-specified and scripted behaviors for advancing on the village. Instead of using scripts for OPFOR behaviors, we created behavior rule sets for the OPFOR to react to significant events in the environment. The initial OPFOR behavior is to harass and delay the advance of the blue forces. OPFOR squad leaders monitor the positions of the blue forces, and initiate a retreat when blue forces cross a particular Line Of Advance. OPFOR squad members monitor the actions of their leaders, and will maintain position or retreat based on their own level of experience, assessment of the environment, and squad leader actions.

We can describe simple versions of the OPFOR behaviors using finite state diagrams. These diagrams are not part of the JCATS interface, but they are useful tools of summarizing the types of state-based rule sets that can be created using the existing JCATS behavior editor (which is text-based). As an example, a simple depiction of the squad leader's behavior is below:

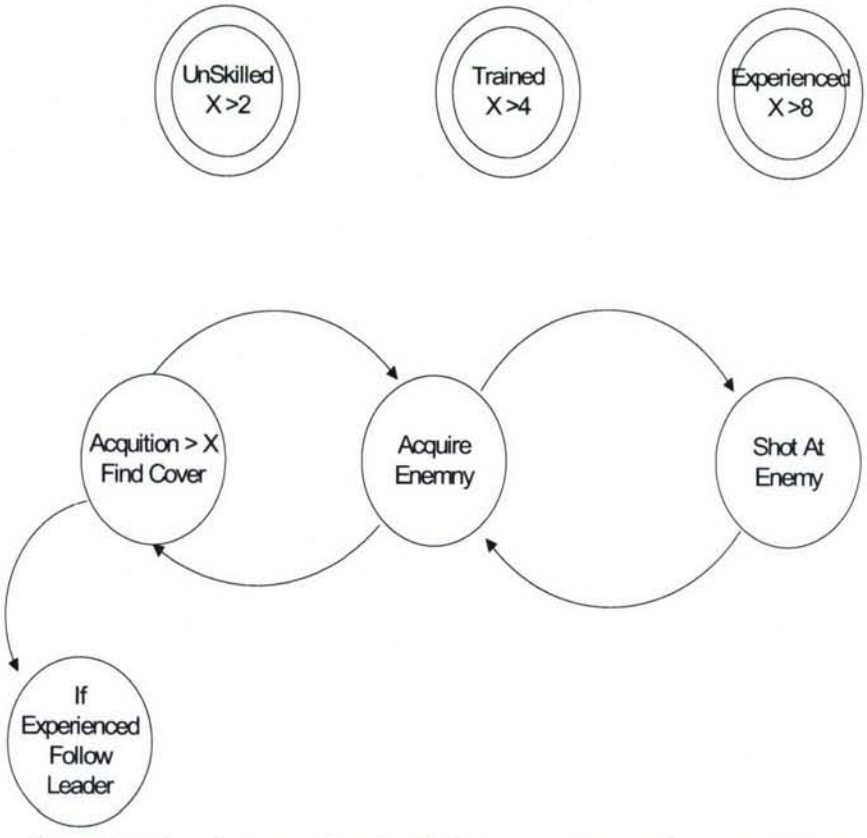


The leader essentially remains in a wait state, observing blue force actions until the blue forces are perceived to be crossing the line of advance. At this point, the leader moves into a new state, where the rules are to select and follow a pre-specified path that will take the leader to a new building, from which to continue the delay. The squad members' behavior contains a few more situations, states and conditions:



Each squad member begins in the Guard state, essentially observing the situation. When a squad member acquires a blue force entity, the squad member changes to the Firing state, and begins shooting at the target. In parallel, the squad member periodically checks the position of the squad leader. If the squad leader is moving, the squad member will change to a Follow state, to maintain the squad's position with the leader. Not depicted here is additional behavior, where the squad member may decide to retreat

(based on level of experience) under different circumstances, even if the squad leader has not moved. The diagram below summarizes this additional behavior:



Each squad member has its own threshold for retreating on its own. The more threatened the squad member feels, the more likely it will begin a retreat. If the squad member acquires any blue forces during the retreat, it will shoot at those forces.

Clearly, these are relatively simple behaviors, so it is a stretch to call these "Thinking" OPFORs. The point of this scenario is to demonstrate basic *types* of behavior that have not typically been used in JCATS. In fact, as we will discuss in the next section, it is very difficult to build any more sophisticated behaviors than this in the existing JCATS behavior editor. But we still able to demonstrate at least rudimentary forms of situational awareness, reacting to changes in the environment, and achieving low-level goals in the context of those changes. This is also the reason we encoded simple forms of levels of experience into the OPFOR models. The idea is to hint at the types of parameterized variability of behavior that we would expect to get out a full-blown force of intelligent entities. Such parameterized models are an excellent way to reduce expense in the deployment of entities. As an example, we can build a single basic rule set governing squad member behavior, and then tailor specifics of the behavior to individual entity parameters that dictate the level of experience (and possibly other factors) associated with each member. The charts below reflect the simple parameterizations we have used for the demonstration scenario, giving different types of reaction and goal-achievement behavior to squad members of different levels of skill based on different assessments of the current situation. Thus, with one basic rule set, plus

the ability to do situation assessment and reaction, the autonomous entities are able to demonstrate a variety of behaviors that enrich the scenario.

Attribute	Case 1	Case 2	Case 3	Case 4
# Enemies	0-2	3-6	7-15	16-Lots
Engage	Yes	Yes	Yes	Yes
Position	Stand	Stand	Crouch	Prone
Speed	Walk	Walk	Run	Crawl

Experienced Skill Matrix

Attribute	Case 1	Case 2	Case 3	Case 4
# Enemies	0	1-3	4-6	7-Lots
Engage	-	Yes	Yes	Hide
Position	Stand	Stand	Crouch	Prone
Speed on Route	Walk	Walk	Run	Crawl

Trained Skill Matrix

Attribute	Case 1	Case 2	Case 3	Case 4
# Enemies	0	1-2	3-4	4-lots
Engage	-	Yes	Hide	Flee
Position	Stand	Crouch	Crouch	Crouch, Prone
Speed on Route	Walk	Crawl	Run	Run

Untrained Skill Matrix

6.0 Assessing Behavior Representation in JCATS

Based on the demonstration scenario, we were able to achieve our goal of creating a reusable set of simple behaviors for future JCATS scenarios. But perhaps more importantly, we achieved the larger technical objective of identifying specific challenges to creating such behaviors in the current JCATS system. In this section, we detail the technical challenges we encountered during the creation of the OPFOR agent behaviors, the short-term solutions we applied to overcome those challenges, and the long-term solutions we would recommend if the JCATS community decided to incorporate a robust autonomous entity behavior capability into the existing JCATS software. The assessments and recommendations we identified fall into five basic categories, which are described below. We present these categories roughly in order of the magnitude of cost it would take to address these challenges within JCATS. That is, the items early on the list include things that we think would be relatively easy to change within JCATS, and would have relatively little impact on other existing JCATS code. The items later on the list are also important to building autonomous entity behaviors, but their solutions would have a higher risk and likely greater impact on existing software.

6.1 Inter-agent Communication

One of the key benefits to autonomous entities is (or ought to be) their ability to communicate and coordinate their actions in teams. Standard procedure for team behavior in JCATS (and other simulation systems) is to have one or more human operators use graphical user interfaces to control the behaviors of individual entities in coordinated ways. Because the coordination really is taking place between human operators, there is no need for communication between the entities themselves, because they are not, in fact, autonomously coordinating with each other. However, as the number of entities increases, and as the desired level of coordination becomes more sophisticated (for example, realistic maneuvering in various formations, with simultaneous reaction to unexpected events), it become onerous even for an experienced group of human operators to generate realistic coordinated behaviors. Thus, having entities that can coordinate on their own is an area that we expect to have some of the biggest cost savings, when such entities can be used in place of human operators.

In order to implement such coordination, however, the entities must be able to communicate with each other. They need to pass situation assessments to each other, leaders need to be able to give orders to subordinates, they need to communicate about progress and achievement of team and individual goals, and in the long run the entities must also be able to communicate with human commanders and operators. Currently in JCATS there is no facility for message passing between entities, because that has just never been a mode of operation in which JCATS entities have been deployed so far.

To work around this limitation in the demonstration scenario, we needed to make use of any mode of communication we could find that *is* currently implemented within JCATS. In the current system, the only type of information that entities can detect from each other is physical presence. That is, for example, a squad member can detect whether its squad leader is currently in sight or not, but has no other way to gather information from the leader. Thus, we used physical presence as a communication signal. The squad leader assesses the situation, and decides whether to begin a retreat. When the leader decides to retreat, it will start moving to a new building. As soon as the squad members

detect that the leader has disappeared, they assume that the leader has begun the retreat, so (depending on level of experience) they will then select their own paths of retreat. This is obviously a very rudimentary form of communication, and is one of the primary reasons that the behaviors in the demonstration scenario are rather simple, while still demonstrating the ability to react to situations and coordinate team behavior.

Clearly, a long-term solution to this challenge is to introduce into JCATS a message-passing facility between entities. This could be as simple as allowing string-based text messages to transmit back and forth, or it could be a more formalized digital representation of some command language. In either event, this would require adding the facility for entities to "activate" or send a message to a particular entity, and it would require a new perceptual facility for entities to receive such messages. There are already sufficient mechanisms in JCATS' behavior rules to generate and process such messages (although those could probably also be improved and tailored to language), so a message-passing facility should be relatively low-risk and low-cost, while also greatly increasing the types of coordinated behaviors that could be implemented.

6.2 Teammate Identification

In the current JCATS model, entities that belong to the same force do not perceive and acquire each other. This is presumably for reasons of efficiency. If there are human controllers supervising the behaviors of all the entities in a single force (which again is currently the standard mode of operation for JCATS), then the human operators will already know the positions of various members on that force. Thus, there is no reason to incur the computational expense of processing the visual acquisition algorithm for all the pair-wise acquisitions that might possibly occur between force members. However, for realistic coordinated behavior in autonomous teams, it is necessary for the entities to have realistic sensing of their force-mates. This is particularly important given the workaround solution we applied to the communication challenge (presented above). We found that the only way to communicate intent between a squad leader and the squad members was for the squad members to detect (visually) whether the squad leader is present. However, this becomes impossible if the squad members cannot ever detect their leader, because the acquisition model doesn't process acquisitions between members of the same force.

Our workaround solution to this problem was to create a separate force for the squad leaders. JCATS allows us to make this separate force and additional "friendly" force (friendly to the OPFOR squad members, that is), but also allows the acquisition algorithm to run. With this workaround, squad members were successfully able to detect the presence or absence of their leaders.

An additional complication, however, is that JCATS does not provide any unique identifying information about entities. When one entity acquires another, the acquisition information includes the force of the acquisition and its force type (such as "scout" or "machine gunner" or "squad leader"), but there is no way to distinguish between individual squad leaders. One possible workaround for this is to make a unique force type of each squad leader, such as "squad leader fred" and "squad leader john", but this quickly becomes onerous.

Long-term solutions to these challenges would allow the acquisition model to run between entities on the same force. Presumably it would be fairly simple to add a flag

allowing this to the existing JCATS code, and doing so would have a small impact on existing code. It would have a potential impact on the efficiency of a running scenario, so it would likely make sense to have this be an optional flag that would only be selected when necessary for autonomous entities. Additionally, we recommend that JCATS adds unique identifying information for entities (such as a call-sign) that can be included in the acquisition information. This would allow a squad member to distinguish between its various squad-mates, as well as between other members of the same force.

6.3 Unrestricted Autonomous Movement

Currently in JCATS, entities can only move independently once they have been directed to follow explicit paths pre-specified by a human operator. The operator does all the reasoning about navigating the terrain, moving around obstacles, and reacting to terrain changes. The operator also can take control of when an entity begins to follow a path, which postures the entity assumes, when it stops, and other aspects of movement. Because all of this is assumed to be controlled by a human, there is no need for the entities even to perceive (in a "cognitive" sense) the terrain. That is, entities will "react" to the terrain in the sense that they will stop moving if they are told to walk through a wall, or similar situations. But the entities have no capability of being "aware" of such terrain features, so they cannot independently make decisions about what to do or where to go.

Full-blown terrain reasoning is a particular human skill that has proven extremely difficult to automate. However, such reasoning is not strictly necessary in a simulated environment. The simulated terrain can be preprocessed and marked with significant attributes, if necessary. This would allow intelligent entities to do certain types of path planning and navigation without having to solve the entire problem of human-level terrain reasoning. However, to implement this capability, the entities must at least be able to sense these terrain markers, such as waypoints. In addition, if entities are to move autonomously and flexibly, they must be able to specify their own paths to follow after they have done route planning through a set of waypoints and landmarks. Similarly, entities must be able to decide and act on their own, when it is necessary to abandon an old path (for example, to react to a change in the environment), assume new postures, change plans, etc.

Our workaround solution for the demonstration scenario was to pre-specify a large number of alternative paths for the individual entities to follow, each with a unique identifier. At run-time, the entities can then do some amount of reasoning to select between these paths, because the JCATS behavior engine provides that capability. However, the entities must know beforehand which paths will be available, and what their unique identifiers are (there is no facility for the entities to "perceive" a new path on the fly…they must know about it ahead of time).

A longer-term solution would build into JCATS new types of sensors, allowing the entities to perceive particular types of landmarks and waypoints associated with the terrain. Additional, new entity actions would allow entities to construct new paths (or to engage in more flexible, non-path-driven movement), and to initiate their own commands about how to execute movements and follow (and abandon) paths. This is a medium-risk, medium-payoff endeavor in JCATS. It is not clear how easy it would be to implement the desired navigation and movement capabilities, but it seems likely that they

could be relatively self-contained within existing path-following modules in the code. In terms of payoff, such changes would make more flexible types of terrain navigation possible, but the existing path-choosing types of behavior may be adequate for many types of autonomous entities.

6.4 Data Structures for Situational Awareness

We have asserted that a key aspect of a "thinking" synthetic agent is the ability to interpret and assess the environment, and maintain sophisticated internal representations of the situation. This assertion relies on the assumption (based on evidence from human reasoning) that it is impossible to generate intelligent behavior without first making some kind of coherent sense out of the environment. This may be particularly true for goal-driven types of behavior, where goals persist even across significant changes in the situation. But even for reactive behavior, such behavior can only be appropriate if the agent is sensible about how it updates its internal awareness model in reaction to environmental changes.

Internal awareness models require some sort of internal state to be represented inside the entities. Any behavior language that supports some type of internal state structures (such as typical variables in compute languages) will be capable of representing internal state in at least a rudimentary form. The current JCATS behavior engine does allow each entity to maintain a set of *user variables*, which can persistently maintain information about the situation. However, the specific design of the data structures available has a large impact on how easy it is to represent different kinds of situations, especially as the situations become more complex. JCATS user variables can hold simple propositional information, such as individual strings of characters or numerical values. However, most intelligent systems provide data structures for representing strongly typed relational information. Such data structure allow for more efficient processing of the data, and also allow behavior designers to use more natural design for the types of information that needs to be stored.

Because JCATS does provide user variables for internal state information, we were able to maintain simple forms of situational awareness representations in the demonstration scenario. However, it is clear that future development and maintenance of more sophisticated "thinking" entities will require more sophisticated types of internal data structures. At the very least, the representational structures should be expanded to support object-oriented types of data that provide strong typing and definitions of relationships between objects. Better would be to provide logical styles of representation, such as Prolog, Lisp, and current rule-based artificial intelligence languages provide. Such data structures could be confined to the behavior module of JCATS, limiting their impact on other portions of the JCATS code. However, it would be a significant effort to build such facilities into the behavior engine, and would have possible implications on the run-time efficiency of JCATS scenarios.

6.5 Data Structures for Goals

Another significant part of the internal state information that an intelligent agent must represent involves its management of goals that drive behavior. Similar to situational awareness, simple goal regimes can be represented by fairly simple state variables. In state-based behavior engines, like that provided in JCATS currently, it is

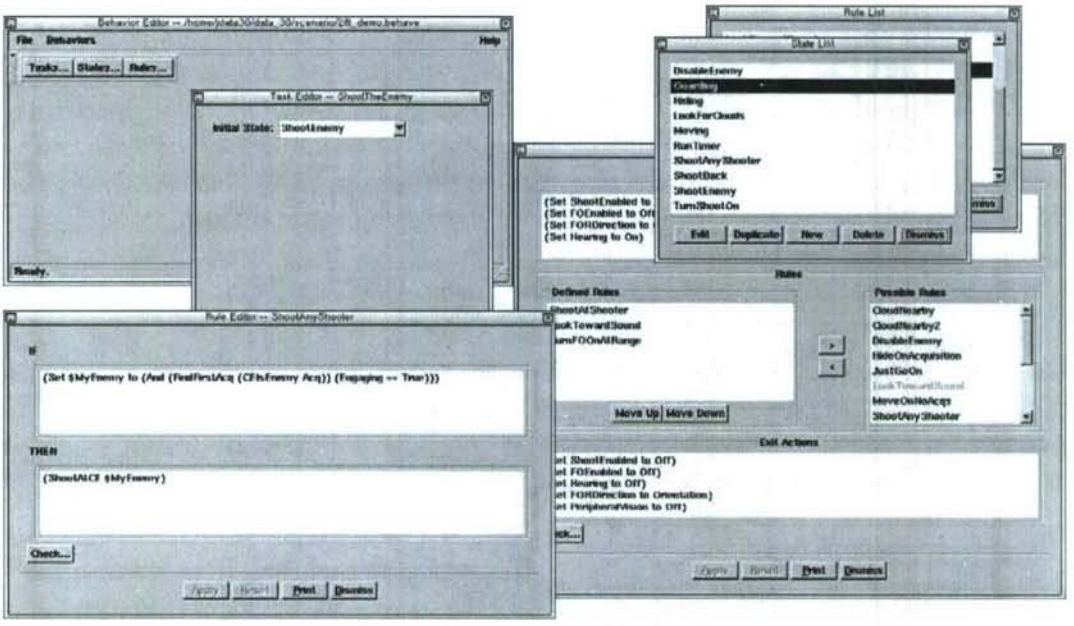
often the case that the "current goal" is stored in the *current state* variable. However, goal information could also be maintained in the user variables that JCATS provides. Thus, for the simple types of autonomous entities we developed for the demonstration scenario, we were able to use existing structures within JCATS to represent goal information. However, more complex agents that reason flexibly in a large range of situations generally require much more sophisticated types of goal structures. Many intelligent agents require the ability to perform hierarchical decomposition on their tasks, and therefore must represent goal hierarchies easily. Additionally, one hallmark of intelligent behavior is the ability to switch between multiple tasks appropriately as fluid situations demand. Thus, it is often necessary for intelligent entities to represent multiple simultaneous hierarchies of active goals, as well as to represent the relationships and constraints between the various goals. This requires representation of relational and hierarchical information that is similar to the requirements for representing situational awareness in complex situations.

Because of these representational requirements, and additional facility for maintaining complex goal structures should be a further long-term requirement for intelligent entities within JCATS. The tradeoffs for these facilities are essentially the same as the tradeoffs for representing situational awareness, since goals are (from one point of view) a functional subset of the situational awareness data structures.

7.0 Evaluation of User Interfaces

As we have suggested, the highest level goals of this project are to determine particular ways for reducing the costs of generating and executing effective scenarios in JCATS simulations. A large part of our effort is directed toward reducing manpower requirements by improving the ability to deploy autonomous entities. However, another method for achieving significant cost savings in any software system is to improve user interfaces that humans use when interacting with the software. Therefore, we also focused our efforts on analyzing existing user interfaces for specifying and executing entity behavior in JCATS, in order to create recommendations and prototypes for improved interfaces.

As with any user interface, the current user interfaces for entity behavior in JCATS are inspired by a particular conceptual model. The conceptual model in JCATS centers on rule-based finite state machines. The basic view of behavior is that, at any specific time, an entity is in a particular *state*. Each state has a set of *rules* associated with it, which are only relevant to behavior while the agent is in that state. Individual rules can make interpretations of sensed information, set the values of user variables, initiate external actions in the simulated world, or cause the agent to change into a new state. The collection of states that an individual agent can navigate defines a *state machine*. Consequently, the graphical editors for developing behaviors in JCATS consist of individual text-based editors for specifying state machines, individual states, and individual rules. The run-time user interface allows a human operator to attach a particular state machine to an entity. A snapshot of the various text-based editors appears below.

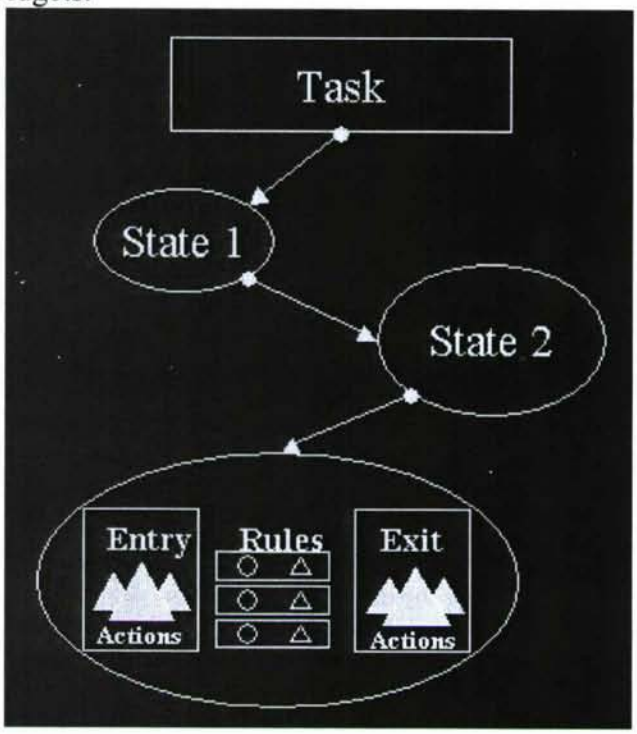


From a usability point of view, using text-based editing tools for each of these tasks does not particularly support the finite-state-machine conceptual model. However, we should emphasize that this mismatch is not particularly surprising, because the behavior editor in JCATS has so far always served essentially as a placeholder for a "behavior implementation to be named later". It was never intended that the current behavior module would provide the ultimate solution for sophisticated entity behaviors in JCATS. However, it is possible to consider the existing tools as a starting point.

In analyzing user interfaces, we again considered two primary alternatives. In the spirit of trying to minimize costs and provide a "good enough" solution, we investigated the possibility of developing new user interfaces that support a computer programmer's conceptual model for building finite state machines. This would be a graphically oriented interface that allows intuitive manipulate of machines, states, and rules. We also investigated a better long-term alternative, which would significantly alter the conceptual paradigm for specifying behaviors, abandoning some of the inflexibility associated with finite state machines.

For both of these analyses, we applied techniques from the field of human-computer interaction. We used a GOMS (Goals, Operators, Methods, and Selections) analysis to create a breakdown of the tasks that a behavior developer must engage in, and combined that with a heuristic evaluation of the current JCATS interfaces and proposed interfaces of our own design. These analyses started by identifying the necessary user actions for specifying a behavior model using the current JCATS behavior engine and user interfaces. We used that identification as a roadmap to drive the heuristic evaluation of cognitive analysis of the thought processes a behavior developer must engage in, and the interfaces available for the developer to translate intentions into action. The results of this analysis provide a set of usability and functionality requirements for a JCATS behavior editor, together with quantitative metrics for evaluating the existing and proposed interfaces.

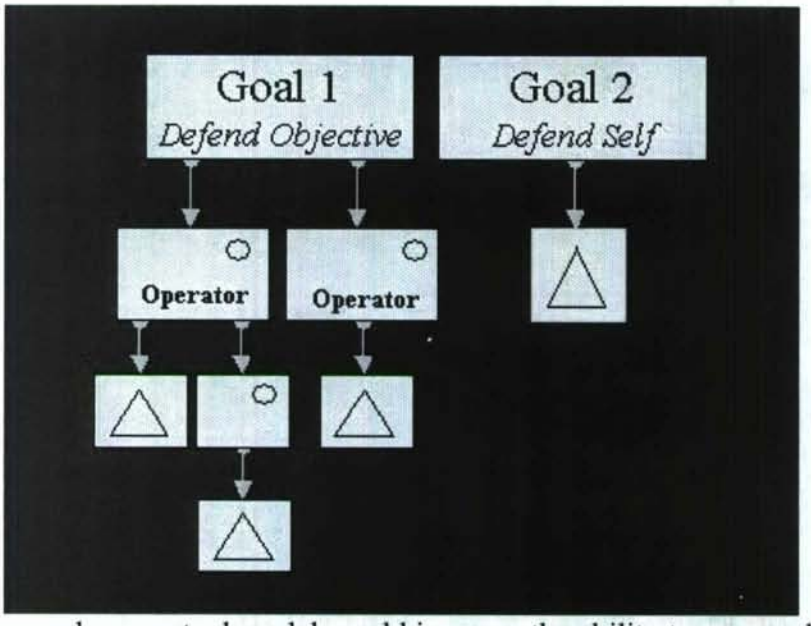
Here we summarize some of the larger lessons learned, and present them with some examples. To begin with, as we have mentioned, the current JCATS behavior editor supports a somewhat complex conceptual model. This model is geared toward software engineers, and not toward subject matter experts. As a result, anybody using these editors must do the work themselves of translating "knowledge level" descriptions of appropriate behaviors down to the finite-state-machine model expected by the behavior editor. Additionally, the developer must further translate concepts at the state-machine level into textual commands in a particular syntax to be entered manually in various text-based widgets.



The need for this multi-layer translation introduces inconsistent approaches to building behavior models, because each individual developer might perform the translation steps differently. It also introduces multiple potential points of failure, at which bugs might be introduced into the design or implementation of the behavior. The conceptual model mixes modes of thought, requiring the developer to address conceptual concerns and engineering concerns simultaneously. Finally, because behaviors are ultimately realized in distributed sets of rules that may be shared across states and states that may be shared across machines, there is poor support of the encapsulation of behaviors at a higher level of abstraction.

Our analysis suggests that the long-term solution is to introduce an improved conceptual model into JCATS for defining entity behaviors. The new model we center on concepts and structures at the task level. This would improve the possibility of having subject matter experts participate in the design of behavior implementations, and would also remove the onus of subjective translation of behaviors by developers into low-level states and rules. The low-level implementation in the new model would be generated automatically by software from higher-level specifications of behavior that focus on

entity goals, intended actions relevant to pursuing those goals, belief maintenance structures and processes, and primitive actions.



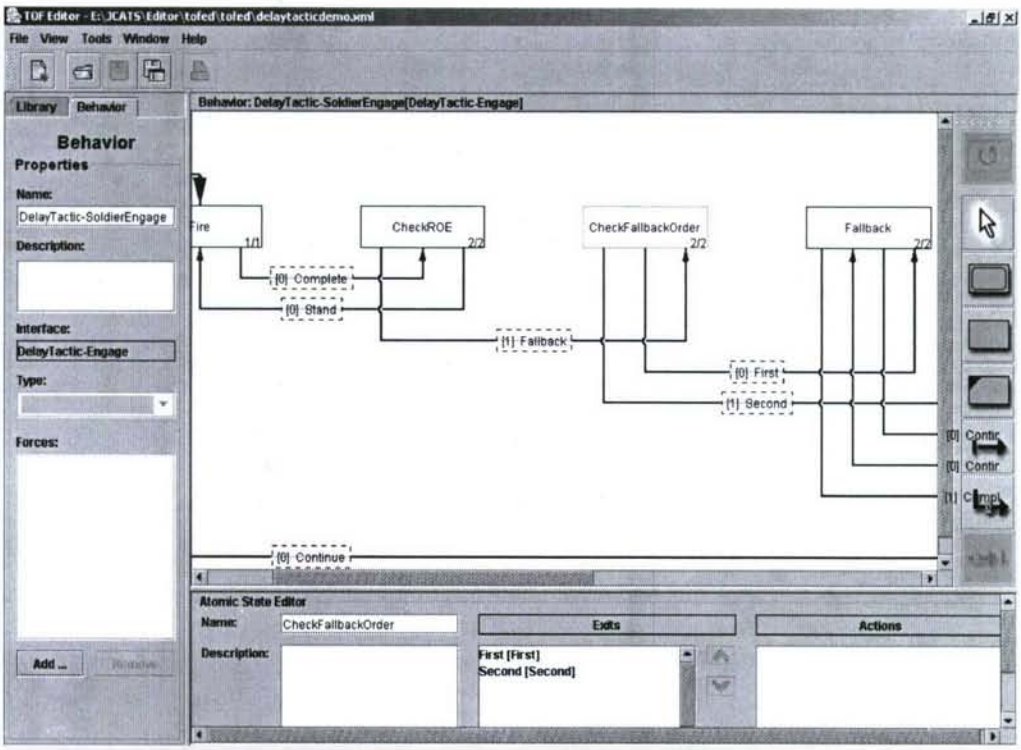
The proposed conceptual model would improve the ability to encapsulate behaviors by allowing them to be expressed in a hierarchically goal-oriented representation language. This would in turn improve the ability to compose behavior units into more complex hierarchical structures. Because goals are first-class entities, they can be represented in a logically distinct fashion from the underlying implementation, and can also be kept logically separate from each other. When an entity needs to reason about multiple simultaneous goals, it can do so by explicitly attending to each goal, instead of embedding implicit goal combinations into the state machine. Most notably from a usability perspective, this conceptual model much more closely matches the types of task decompositions that users and subject matter experts use when thinking about behavior. This reduces the number of steps necessary to translate from task-level descriptions to implementation-level details.

An advantage of this conceptual model is that it could, in theory, map to the existing state-based model, and this mapping could be encoded in automated tools for translating behavior representations. However, the new model would also support specifications of behavior for other types of behavior engines, such as modern cognitive architectures for building human-like agent systems.

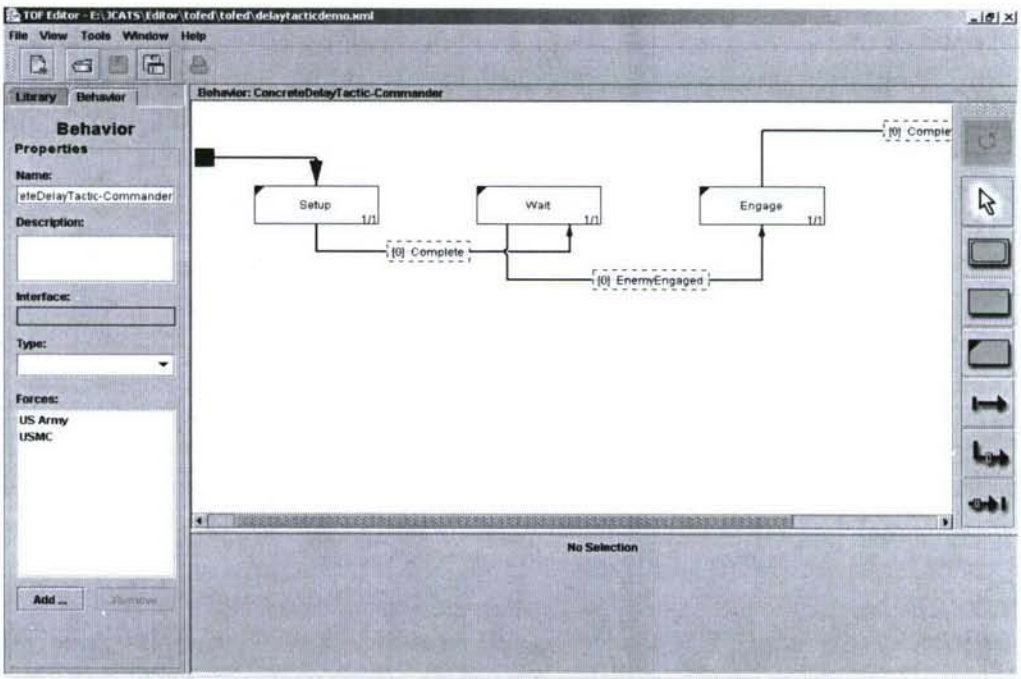
We used this new conceptual model, together with our usability analysis of the state-based model, to design prototype behavior development tools that could be used in the existing JCATS behavior editor, or in more sophisticated future editors. The new editor begins with a graphical editor for high-level "Behaviors". "Behavior" descriptions provide a detailed specification of the actions for a single entity that can be combined to form "Orders".

An additional screen allows "Orders" to be composed from individual behaviors. "Orders" are high-level conceptual objects that focus on mission objectives, as opposed to particular sub-tasks of missions. Orders can be defined as state machines, where each state is itself an abstract behavior. The behaviors are more detailed state machines. In

addition, the orders are displayed in the environmental context in which they are to be executed, which can aid the developer in debugging the behaviors, or can also aid users in validating the behaviors.

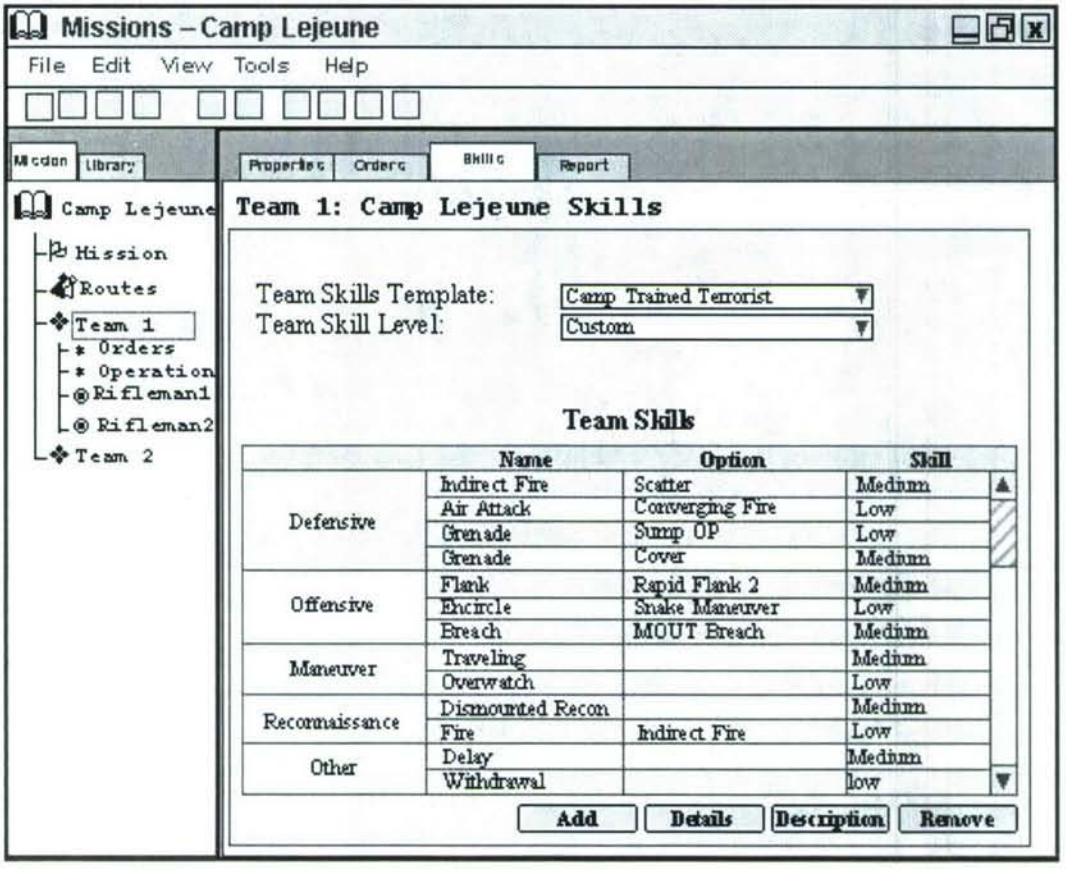


Prototype Behavior Editor



Prototype Orders Editor

Additional prototype behavior editors allow developers and users to specify coordinated actions for teams of agents, as well as assigning team roles, mixtures of experience levels, and other team-oriented features. The "Team Skills" interface allows to composition of team behaviors from previously defined individual behaviors. It allows team behaviors to be defined from previously constructed templates that include specifications for an appropriate mix of skill levels, or it can allow teams to be composed by custom selection of individual parameter settings for the team members.

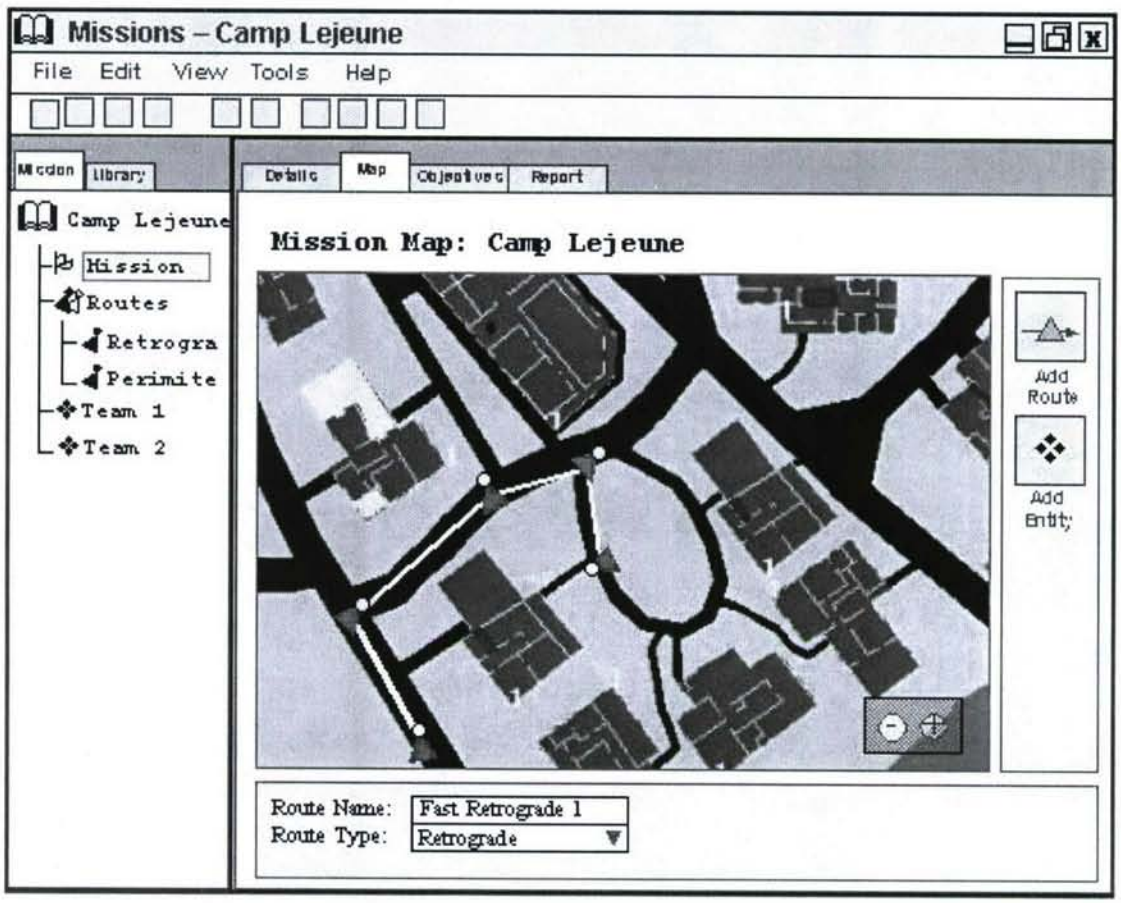


Team Skills Editor

Finally, an integrated map-based editor can allow initial placement of forces, together with visual planning of force routes, navigation, and other activities. In the current JCATS system, the map-oriented planning interface is completely distinct from the behavior editor. However, it seems clear that behavior specifications will often refer to map elements and elements of the terrain. Thus, to maximize ease of use, we propose a graphical map editor that is tightly integrated with the other behavior editing tools.

8.0 Project Recommendations

After completing our analyses, demonstrations, and prototypes, we held a meeting with the JCATS developers at Lawrence Livermore, to discuss our proposed alternatives and assess the technical feasibility of various approaches to enhancing JCATS support for autonomous entities. These discussions produced a clear technical recommendation that we subsequently presented to the JCATS configuration control board, constituted of the primary user communities for JCATS.



Integrated Map Editor for Behaviors

Regarding the software engine for producing autonomous entity behavior, we judged that integrating enhancements into the existing JCATS code base would be of prohibitively high risk and cost. The potential benefits of such an approach would be to ensure a uniform presentation of interfaces and processes for the various capabilities that JCATS provides. The primary disadvantage would be that such an approach would break modularity. Essentially, we would be taking a very well engineered simulation engine, and forcing it simultaneously to become a very well engineered autonomous behavior engine. These are conceptually distinct facilities (although they ultimately need to integrate together). Good software engineering practices dictate that we keep these components separate as much as possible, so each can perform their tasks as efficiently and conceptually cleanly as possible.

Thus, our strong recommendation (together with the JCATS developer) is that the JCATS development path pursue the alternative of building a separate "behavior server" that would network with JCATS servers and clients to provide desired autonomous entity behaviors. This approach has numerous advantages:

- It would minimize "reinventing the wheel", by trying to duplicate years of effort in building intelligent agent architectures within the existing JCATS code. There are several sophisticated cognitive architectures and intelligent agent architectures that could serve as the basis for the behavior server.
- It would minimize intrusion into the existing JCATS code base, thus greatly reducing the risk of introducing new bugs into the mature JCATS code. This would allow software maintenance of the behavior server and other JCATS components to proceed relatively independently, which is nearly always the correct approach for long-lived, large, and complex software systems.
- It would minimize the run-time impact of the behavior engine on existing operations with JCATS. Because the JCATS simulation and the behavior simulation would occur in different processes (and possibly in most applications on different machines, networked together), the behavior server would not impact the run-time efficiency of the rest of the simulation. It is likely that the behavior server *would* generate additional network traffic, which could slow down some simulation processes. But, because of the separation of the two, it would be an easy matter to disconnect the behavior server from the network in those cases where thinking types of entities are not required. Thus, the behavior server alternative provides an optional behavior capability that does not have to have an impact on legacy applications of JCATS.

9.0 Deliverables

This project has produced a number of deliverable items, which we enumerate and describe here. We have attempted to produce a variety of deliverables to maximize the potential payoff of our efforts. Our desire is that the primary conclusions and recommendations of this project will be used to justify efforts to create a behavior server and integrated behavior editing tools for JCATS. We believe such tools would also be reusable in other simulation environments. However, it is worth pointing out that there may currently be no desire in the JCATS community for the sophisticated level of entity behaviors that have been the subject of our investigation. In that case, the expense of creating a behavior server for JCATS would likely not be justified. But we hope that the other deliverables we have produced, such as documentation, analyses, prototypes, and examples, will prove useful for operators and behavior developers even if there is no future work on enhancing the JCATS behavior engine and editors.

1. A "Thinking OPFOR" Graphical Editor. We completed construction of a prototype editor for autonomous entity behaviors, based on usability analyses of behavior construction tasks. Supporting this prototype, we have also generated documentation describing the analysis and usability of the existing JCATS editor. The prototype does not actually generate JCATS behavior files, but we have documented how it can be made to do so. The prototype does generate behavior specifications in XML, which

should serve as a sufficiently generic specification language for translating behaviors into JCATS or other types of behavior engines.

2. A Demonstration Rule Set for "Thinking OPFOR" Behavior. Using the existing JCATS behavior tools (not our prototypes), we generated autonomous entity behavior sets within the demonstration MOUT scenario that we used for our studies. Due to the limitations of the current JCATS behavior engine, these behaviors are not as sophisticated as we originally hoped they would be. However, even given the modest amount of autonomy they exhibit, these behaviors are still significantly different from the entity behaviors usually used in JCATS. Standard JCATS scenarios require human operators to interpret, assess, and control the entities. The rule sets we developed at least allow a small degree of autonomy, including simple assessments of (and reaction to) the situation, and simple versions of behavior coordinated in teams. The behaviors run in the existing JCATS simulation, and are parameterized using various "skill levels" for the OPFOR teammates. Thus, the behavior files should provide a modest reusable library for operators who want to incorporate them into their own JCATS scenarios. We have also generated documentation and graphical depictions of the behavior set, to add future operators in understanding what we have generated.
3. JCATS Portion of Field Test Plans. This document describes the testing plan we undertook to evaluate the current JCATS behavior engine and associated user interfaces.
4. Baseline Scenario Translated into JCATS. The baseline scenario for our prototype demonstration of autonomous entities was designed and translated into JCATS format primarily by our project partners at Alion. We made additional enhancements to the JCATS implementation, and the resulting JCATS specification files can be reused by anyone who wants to run them within JCATS.
5. Excursion Scenario Translated into JCATS. Our original intent was to demonstrate two specific scenarios within JCATS. As conditions changed, we decided on combining the two scenarios into a single scenario that highlights the technical objectives of both. Thus, the originally planned "baseline" and "excursion" scenarios are compressed into a single scenario that serves now as our prototype demonstration scenario.
6. Final Version of JCATS Portion of User's Manual. While performing our evaluations of the existing behavior editing tools within JCATS, we also generated user documentation to make it easier for developers to specify entity behaviors within the existing JCATS behavior engine. We feel this documentation provides more useful information than the behavior documentation currently incorporated into JCATS.

10. Unanticipated Factors

We feel we have made a substantial contribution to knowledge about autonomous entity behavior, particularly with regard to the JCATS simulation system, and we feel that we have done so by meeting all of the requirements of the contract for this project. However, as with any research project of sufficient size, we did run into some roadblocks that resulted in various adjustments to our original plans. Some of the technical obstacles and work-around are described earlier in this report, but we feel it is worth describing some of the remaining roadblocks and adjustments here.

To begin with, this project demanded much more travel than originally planned. This travel was extremely useful, because it allowed us close contact with various members of the JCATS user community, in order to refine the direction of our research, and to help us ensure we were producing results that would be useful to the community. However, travel does impose a cost, and this project had a limited amount of funds. Therefore, we did have to trade off some of our originally planned efforts to make up for the additional travel costs we incurred.

One of the primary changes in this regard was to reorganize the demonstration scenario we constructed within JCATS. Our original intent was to construct a set of demonstration scenarios highlighting various levels of autonomous behavior. Partly in response to the need to cut costs, we changed plans to instead provide a single demonstration scenario that would exhibit a variety of different types of autonomous behavior. We attempted to compensate in part for this change by expanding our efforts to model a variety of skill levels in the team member behaviors. Thus, we think this change overall had a benefit, because it encouraged us to do an even better job of building a parameterized behavior rule set that can be reused to generate various levels of behavior. While we were doing this reimplementation of the original scenarios, we encountered a further unexpected quirk in JCATS' simulation of aggregate entities. It turns out that aggregate units within JCATS share perceptual information with each other, so they are sometimes more powerful as an aggregate than they would be if they were represented as individual units. In order to eliminate this unrealistic advantage (thereby keeping the scenario interesting), we retooled the entire original scenario to replace the aggregate blue force units with an equivalent (and large) number of individual JCATS units.

11. Remaining Questions

This project has prepared groundwork and produced results that could be taken in a variety of different directions, depending on future levels of interest and funding. We feel the primary remaining question to determining where to allocate future resources is whether there would be a significant operational benefit to adding autonomous entities to existing simulation systems such as JCATS. We have attempted to argue the potential long-term cost savings of including such agents, and there are other simulation systems that have successfully incorporated autonomous agents of varying levels of competence. However, the simulation user community must ultimately answer the question, after being informed by the technical community with the types of studies we have completed here. It would be beneficial to extend this work by analyzing current and (hypothetical) future user needs for simulation, as well as existing costs for developing and deploying scenarios. Based on those results, it would be useful to revisit the question of the value

of autonomous entities, and then possibly use the results of this project to achieve whichever goals are deemed most useful for simulated entity behavior.

Related to the emphases of this project, an additional question is to evaluate the operational benefit of improved user interfaces for specifying entity behavior, including entity planning and control. Even if the current technology for entity behavior in JCATS is adequate, it seems clear that improved user interfaces would decrease the cost of creating and deploying entity behaviors, and probably also increase the frequency of their use.

12. Courses of Action

Based on the results of our research and an assessment of the answers to the above questions, it may be useful to enumerate potential courses of action from this point. The first obvious potential choice is to maintain the status quo. It is clear that the current version of JCATS is popular, and serves a number of customer needs. Perhaps there is no significant benefit to improving entity behavior or user interfaces for the types of tasks that JCATS is currently used for. Additionally, depending on the amount of use new behaviors and interfaces would get, it may not justify the expense of investing in their development. We feel this is probably not true, but again the true answer may not be known without a more formal assessment of customer needs, requirements, preferences, and expenses. By maintaining things as they are, it is still possible to generate some limited types of autonomous entities.

A somewhat conservative alternative would be to maintain the existing behavior engine in JCATS, but to supplement the user interfaces with new ones based on our analyses and prototypes (or of some other improved design). The advantage to this approach is that there would be relatively small cost to introducing the new user interfaces, and they would allow cheaper development of simple behavior models. Also, adding these new user interfaces would consist of incremental code changes that could be folded into the existing JCATS development cycle.

A third option would be to restructure the current JCATS behavior system significantly, following the recommendations we generated to overcome current technical obstacles to generating sophisticated entity behaviors. Of course, this option should only even be considered if there is judged to be a value to having smarter autonomous entities in JCATS simulation. But even if that value exists, we would recommend against this option, for the reasons we have outlined earlier in this report. We include the option here in the report summary for the sake of completeness. The benefit of this approach is that it would provide the infrastructure for sophisticated autonomous agents. But the disadvantages are that this approach would significantly impact the current JCATS development cycle, and the ultimate solution would then be tied to JCATS, limiting the potential for a behavior module that could be reused in other simulation systems.

A fourth option, and the option we recommend if thinking synthetic entities prove to be desirable to the user community, is to create an independent "behavior serve" that communicates with JCATS servers and clients using the JCATS network protocol. Again, we have described this approach and its advantages previously in this report, but we summarize here. The primary disadvantage to this option is that it would likely involve a medium to high cost investment to complete such a project. However, the benefits would be worth it if we are correct about the cost-saving value of autonomous

entities. This approach would build on existing systems for building cognitive and intelligent agents, thus providing infrastructure for building entities that can have very complex and sophisticated behaviors. In addition, although the behavior server would include a component for communicating with JCATS, the behavior engine itself would essentially be independent of the details of the JCATS simulation environment. This opens up the potential for reusing the behavior server in other types of simulation systems, and also for using the behavior server in heterogeneous simulation environments, perhaps networking JCATS with other simulation engines.